01_simus_manual

January 28, 2018

1 Probando PuLP

1.1 Reproducción del Paper: SIMUS UN MÉTODO DE PROGRAMACIÓN MULTIOBJETIVO

Munier, N., Carignano, C., & Alberto, C. UN MÉTODO DE PROGRAMACIÓN MULTIOBJETIVO. Revista de la Escuela de Perfeccionamiento en Investigación Operativa, 24(39).

1.1.1 Enunciado

A fin de mostrar una aplicación del método, se utiliza un ejemplo 48 SECCION APLICACIONESINVESTIGACION OPERATIVA - AÑO XXIV - No 39 - PAGINAS 44 a 54 - MAYO 2016 sobre planificación urbana, propuesto en Lliso (2014) 2 .

Desde hace algunas décadas y debido a los cambios en la industria del transporte ferroviario y marítimo en casi todas las ciudades hay espacios vacíos, por lo general céntricos, los cuales eran ocupados por las estaciones de ferrocarril, patios de ferrocarril y muelles. Estas parcelas abandonadas son el blanco de los municipios para ser usadas en la construcción de parques, edificios de viviendas, oficinas gubernamentales, centros comerciales, etc. Suelen ser grandes parcelas que pueden ser adecuadas para varios usos diferentes al mismo tiempo, razón por la cual el gobierno municipal se encuentra ante un dilema ya que es deseable seleccionar la alternativa de uso que mejor sirva a la ciudad.

CASO: REHABILITACIÓN DE TIERRAS Una importante ciudad portuaria se ha visto afectada por el cambio en la modalidad de transporte marítimo, cuando comenzó el transporte de contenedores a mediados del siglo 20. La ciudad se quedó con 39 hectáreas de muelles vacíos, así como los almacenes y una terminal ferroviaria.

El municipio tiene que decidir qué hacer con esta tierra y desarrolló un Plan Maestro basado en tres proyectos diferentes:

- Proyecto 1: Torres Corporativas - Hoteles - Marina - Pequeño Parque.
- Proyecto 2: Torres de Viviendas - Zona Comercial en la antigua estación de ferrocarril.
- Proyecto 3: Centro de Convenciones - Gran Parque y área recreativa.

Los criterios que considera para el análisis de las propuestas se refieren a:

- Criterio 1: Generación de Trabajo
- Criterio 2: Espacio Verde Recuperado

- Criterio 3: Factibilidad Financiera
- Criterio 4: Impacto Ambiental

Para el criterio 2 se especifica un límite máximo, mientras que los restantes criterios no tienen restricción definida para el lado derecho.

El Decisor considera a los cuatro criterios como objetivos, por lo que se deberán resolver cuatro programas lineales con tres restricciones cada uno.

Los datos se detallan en la tabla siguiente.

	Proyecto 1	Proyecto 2	Proyecto 3	VLD	Acción
Criterio 1	250	130	350	-	Maximizar
Criterio 2	120	200	340	500	Maximizar
Criterio 3	20	40	15	-	Minimizar
Criterio 4	800	1000	600	-	Maximizar

In [1]:
```
import pulp

import numpy as np

from skcriteria import norm
```

Stage 1

In [2]:
```
stage1 = pulp.LpProblem("Stage 1", pulp.LpMaximize)

# variables
x1 = pulp.LpVariable('x1', lowBound=0, cat='Continuous')
x2 = pulp.LpVariable('x2', lowBound=0, cat='Continuous')
x3 = pulp.LpVariable('x3', lowBound=0, cat='Continuous')

# Objective
stage1 += 250 * x1 + 130 * x2 + 350 * x3, "Z"

# Constraints
stage1 += 120 * x1 + 200 * x2 + 340 * x3 <= 500
stage1 += 20 * x1 + 40 * x2 + 15 * x3 >= 15
stage1 += 800 * x1 + 1000 * x2 + 600 * x3 <= 1000

stage1
```

Out[2]:
```
Stage 1:
MAXIMIZE
250*x1 + 130*x2 + 350*x3 + 0
SUBJECT TO
_C1: 120 x1 + 200 x2 + 340 x3 <= 500

_C2: 20 x1 + 40 x2 + 15 x3 >= 15
```

```
_C3: 800 x1 + 1000 x2 + 600 x3 <= 1000

VARIABLES
x1 Continuous
x2 Continuous
x3 Continuous
```

Stage 2

In [3]:
```
stage2 = pulp.LpProblem("Stage 2", pulp.LpMaximize)

# variables
x1 = pulp.LpVariable('x1', lowBound=0, cat='Continuous')
x2 = pulp.LpVariable('x2', lowBound=0, cat='Continuous')
x3 = pulp.LpVariable('x3', lowBound=0, cat='Continuous')

# Objective
stage2 += 120 * x1 + 200 * x2 + 340 * x3, "Z"

# Constraints
stage2 += 250 * x1 + 130 * x2 + 350 * x3 <= 350
stage2 += 20 * x1 + 40 * x2 + 15 * x3 >= 15
stage2 += 800 * x1 + 1000 * x2 + 600 * x3 <= 1000

stage2
```

Out[3]:
```
Stage 2:
MAXIMIZE
120*x1 + 200*x2 + 340*x3 + 0
SUBJECT TO
_C1: 250 x1 + 130 x2 + 350 x3 <= 350

_C2: 20 x1 + 40 x2 + 15 x3 >= 15

_C3: 800 x1 + 1000 x2 + 600 x3 <= 1000

VARIABLES
x1 Continuous
x2 Continuous
x3 Continuous
```

Stage 3

In [4]:
```
stage3 = pulp.LpProblem("Stage 3", pulp.LpMinimize)

# variables
x1 = pulp.LpVariable('x1', lowBound=0, cat='Continuous')
```

```
x2 = pulp.LpVariable('x2', lowBound=0, cat='Continuous')
x3 = pulp.LpVariable('x3', lowBound=0, cat='Continuous')

# Objective
stage3 += 20 * x1 + 40 * x2 + 15 * x3, "Z"

# Constraints
stage3 += 250 * x1 + 130 * x2 + 350 * x3 <= 350
stage3 += 120 * x1 + 200 * x2 + 340 * x3 <= 500
stage3 += 800 * x1 + 1000 * x2 + 600 * x3 <= 1000

stage3
```

Out[4]:
```
Stage 3:
MINIMIZE
20*x1 + 40*x2 + 15*x3 + 0
SUBJECT TO
_C1: 250 x1 + 130 x2 + 350 x3 <= 350

_C2: 120 x1 + 200 x2 + 340 x3 <= 500

_C3: 800 x1 + 1000 x2 + 600 x3 <= 1000

VARIABLES
x1 Continuous
x2 Continuous
x3 Continuous
```

Stage 4

In [5]:
```
stage4 = pulp.LpProblem("Stage 4", pulp.LpMaximize)

# variables
x1 = pulp.LpVariable('x1', lowBound=0, cat='Continuous')
x2 = pulp.LpVariable('x2', lowBound=0, cat='Continuous')
x3 = pulp.LpVariable('x3', lowBound=0, cat='Continuous')

# Objective
stage4 += 800 * x1 + 1000 * x2 + 600 * x3, "Z"

# Constraints
stage4 += 250 * x1 + 130 * x2 + 350 * x3 <= 350
stage4 += 120 * x1 + 200 * x2 + 340 * x3 <= 500
stage4 += 20 * x1 + 40 * x2 + 15 * x3 >= 15

stage4
```

Out[5]:
```
Stage 4:
MAXIMIZE
```

```
800*x1 + 1000*x2 + 600*x3 + 0
SUBJECT TO
_C1: 250 x1 + 130 x2 + 350 x3 <= 350

_C2: 120 x1 + 200 x2 + 340 x3 <= 500

_C3: 20 x1 + 40 x2 + 15 x3 >= 15

VARIABLES
x1 Continuous
x2 Continuous
x3 Continuous
```

1.1.2 Solutions

```
In [6]: stages = [stage1, stage2, stage3, stage4]
```

```
In [7]: for stage in stages:
            stage.solve()
            status = pulp.LpStatus[stage.status]
            print(f"{stage.name}: {status}")
```

```
Stage 1: Optimal
Stage 2: Optimal
Stage 3: Optimal
Stage 4: Optimal
```

```
In [8]: results = []
        for stage in stages:
            row = [v.varValue for v in stage.variables()]
            results.append(row)


        results = np.asarray(results)
```

Normalization

```
In [9]: renorm = norm.sum(results, axis=1)
        renorm[np.isnan(renorm)] = 0
        renorm
```

```
/home/juan/proyectos/skcriteria/src/skcriteria/norm.py:173: RuntimeWarning: invalid value encoun
  return arr / sumval
```

```
Out[9]: array([[0.125     , 0.        , 0.875     ],
               [0.        , 0.38888889, 0.61111111],
               [0.        , 0.        , 0.        ],
               [0.05681818, 0.94318182, 0.        ]])
```